A Genetic Algorithm for Disassembly Process Planning

Elif Kongar; Surendra M. Gupta

Kongar, E. and Gupta, S.M., “Genetic Algorithm for Disassembly Process Planning”, Proceedings of the SPIE International Conference on Environmentally Conscious Manufacturing II, Newton, MA, October 28-29, pp.54-62, 2001.

doi: http://dx.doi.org/10.1117/12.455264

A Genetic Algorithm for Disassembly Process Planning

Elif Kongar and Surendra M. Gupta*
Laboratory for Responsible Manufacturing
334 SN, Department of MIME
Northeastern University
360 Huntington Avenue
Boston, MA 02115.

ABSTRACT

When a product reaches it's end-of-life, there are several options available for processing it including reuse, remanufacturing, recycling, and disposing (the least desirable option). In almost all cases, a certain level of disassembly may be necessary. Thus, finding an optimal (or near optimal) disassembly sequence is crucial to increasing the efficiency of the process. Disassembly operations are labor intensive, can be costly, have unique characteristics and cannot be considered as reverse of assembly operations. Since the complexity of determining the best disassembly sequence increases with the increase in the number of parts of the product, it is extremely crucial that an efficient methodology for disassembly process planning be developed. In this paper, we present a genetic algorithm for disassembly process planning. A case example is considered to demonstrate the functionality of the algorithm.

Keywords: Disassembly Scheduling, Disassembly Process Planning and Genetic Algorithm.

1. INTRODUCTION

Today, environmental consideration is a necessity for responding to the decreasing number of landfills and virgin resources. Increasing number of governments are holding the manufacturers responsible to take-back their products at the end of their lives. In response, the manufacturers are seeking solutions to address the potential accumulation of large inventories. This inventory may consist of outdated and/or non-functioning products. As far as the electronic products are concerned, which are the focus of this paper, the rapid pace of technological development has the potential to render the products obsolete after a short time, even though most of them may be in excellent working conditions. The challenge is to process these products in an environmentally benign and cost effective manner. There are various ways to accomplish this task under the general umbrella of end-of-life (EOL) processing. Some of the options for EOL processing include reuse, recycle, storage for future use or disposal. Even though the disposal option is not desirable, there are times when it is the only option available. In such cases, the method of disposal chosen should be the one that would be least harmful to the environment.

In majority of EOL processing, a certain level of disassembly may be necessary. Even in the disposal option, the hazardous contents are separated from the product and carefully processed before the residual product is disposed of. *Disassembly* is the process of systematic removal of desirable constituents (components and/or materials) from the original assembly so that there is no impairment to any useful constituent. Disassembly can be *partial* (product not fully disassembled) or *complete* (product fully disassembled) and may use a methodology that is *destructive* (focusing on materials rather than components recovery) or *non-destructive* (focusing on components rather than materials recovery). In this paper, we consider the case of complete disassembly where the components retrieved are by either destructive or non-destructive methodology.

Many products are made up of a large number of components. The optimal disassembly path to retrieve the components could always be obtained using exhaustive search algorithms. However, because of the combinatorial nature of the problem, as the number of components in a product grow, the number of possible disassembly combinations grows exponentially, making the path selection practically prohibitive from the point of view of time required and cost. For this reason, heuristic methods are often employed to find near-optimal solutions. These approaches can frequently provide satisfactory results

*Correspondence: e-mail: gupta@neu.edu; URL: http://www.coe.neu.edu/~smgupta/
Phone: (617)-373-4846; Fax: (617)-373-2921

Environmentally Conscious Manufacturing II, Surendra M. Gupta, Editor,
Proceedings of SPIE Vol. 4569 (2002) © 2002 SPIE · 0277-786X/02/$15.00

much faster and at low costs. In this paper, we present a genetic algorithm for the disassembly process as described above in the presence of constraints and precedence relationships.

2. LITERATURE REVIEW

In recent years, genetic algorithm (GA) has been gaining in popularity as a tool of choice to solve combinatorial problems or problems that grow exponentially in complexities as their sizes grow. GAs are heuristic techniques and can provide quick and cost effective solutions to problems that would otherwise take prohibitive amount of time. The price one has to pay is in terms of the type of solution one gets. Even though by using GA an optimal solution is not always guaranteed a reasonable (and in many cases optimal) solution is often obtained and thus offers a good compromise for a large class of problems including disassembly processing.

Valenzuela-Rendón and Uresti-Charre (1997) proposed a non-conventional GA for multi-objective optimization and concluded that the described GA had more stable and reliable time response. Loughlin and Ranjithan (1997) also proposed a GA method, a so called neighborhood constraint method, and concluded that the GA performed better in multi-objective problems compared to single objective problems.

One of the most popular areas where GA has been applied is in sequencing and scheduling problems. Recently, Seo *et al.* (2001) proposed a genetic algorithm for generating optimal disassembly sequence considering both economical and environmental aspects. Their algorithm aimed to obtain the optimal disassembly sequence. However their search could lead to infeasible strings during the crossover and mutation operations. The authors addressed this situation by penalizing the string with the hope that it would be eliminated during latter generations. Therein lies the weakness in the algorithm.

Lazzerini and Marcelloni (2000) also used GA in scheduling but for strictly assembly processes. They used modified partially matched crossover (PMX) and mutation operations to obtain the near optimal sequence. The precedence relationships were not considered in the model.

Along with the speed, ease and low cost, there are other reasons to use heuristic search algorithms. GAs may find optimal results to problems, which cannot be solved using linear-integer programming or similar techniques. This being the motivation, Onwobulu and Mutingi (2001) used GA for optimizing a product mix problem under multiple constrained resources. The authors reported that the results were encouraging supporting the use of the algorithm.

It is no longer possible to deny the importance of EOL processing. Today the manufacturers are receptive to considering the recovery concept, whatever the reason may be. Ritchey *et al.* (2001) recently analyzed the economics of the remanufacturing processes. Wong (2001) investigated the difficulties and main problems in EOL activities.

Various researchers have studied disassembly, being one of the primary elements for parts and product recovery. Kuo (2000) analyzed the cost of disassembly in electromechanical products. Taleb and Gupta (1997) studied disassembly in multiple product structures. Later Veerakamolmal *et al.* (1997) improved the methodology and applied this approach to a multi-period disassembly environment. Moore *et al.* (2001) handled the subject a bit differently. The authors used Petri-nets to study products with complex AND/OR relationships. More detailed overviews of such systems are given by Moyer and Gupta (1977), Gungor and Gupta (1999), Tang *et al.* (2000) and Lee *et al.* (2001).

Precedence relationships are one of the factors that add to the complications in sequencing problems. The conventional search algorithms generally use combinatorial search techniques and sort of augment them with precedence relationships. Sanderson *et al.* (1990) considered precedence relationships in assembly sequence planning in such a manner. Similarly, regular GAs are generally not suitable for the systems where precedence relationships and constraints are involved. Bierwirth *et al.* (1996), Bierwirth and Mattfeld (1999) proposed a methodology to overcome this problem by introducing the precedence preservative crossover (PPX) technique for scheduling problems. Blanton and Wainwright (1993) first introduced the basic methodology for the vehicle routing problems. The methodology preserves the precedence relationships during the crossover function of GA. The method guarantees feasible results at each of the steps.

3. A GENETIC ALGORITHM FOR THE DISASSEMBLY PROCESS

The idea behind the genetic algorithm was inspired by Darwin's theory of evolution. The algorithm starts with a *set of solutions* called the *population. Chromosomes* represent each solution in the population. Solutions from one population are taken and used to form a new population. The hope is that the new population will be better than the old one. Solutions that are selected to form new population, or *offspring*, are chosen according to their fitness. This is repeated until some predetermined condition is satisfied. Note that in our proposed version of genetic algorithm, the feasibility of each disassembly solution (i.e. the precedence relationship) is always preserved starting from the initial population until the final population is generated.

In the following subsections we describe the elements and development of a genetic algorithm with the help of an example as it pertains to disassembly.

3. 1. Genetic Representation

Chromosomes

The representation of each chromosome (solution) in the population consists of three parts of equal length. The first part represents the disassembly sequence of components in the product. For example, the product structure given in Fig. 1 consists of ten components represented by integers from 0 to 9. The second part of the chromosome represents the direction in which the disassembly is to be performed. The direction is shown in Table 1 and consists of coordinates *x, y, z* and *-x, -y, -z*. The last part of the chromosome indicates whether the methodology used for disassembly is destructive or non-destructive. An example of the chromosome structure is as follows:

Chromosome: 2 7 0 6 4 1 5 3 8 9, $+x\ -x\ +y\ +y\ -z\ +x\ -z\ +y\ -z\ -y$, $N\ \mathbf{D}\ D\ D\ N\ N\ N\ D\ N\ D$

As can be seen from the above example, component 7 is to be disassembled after component 2 and prior to component 0 and requires a disassembly in the $-x$ direction and is subjected to destructive (D) disassembly. Note that the letter D represents destructive disassembly while the letter N represents non-destructive disassembly.

Initial population

The initial population consists of n chromosomes. A repetitive random selection is performed to generate the n chromosomes. This is done such that all the precedence relationships and any other constraints imposed by the product structure are satisfied. As an example consider the product in Fig. 1. In this example the precedence relationships are as follows: component 1 **or** 2 must be disassembled prior to any other component; component 6 must be disassembled prior to components 4 **and** 5; and component 7 must be disassembled prior to components 6 **and** 3. Table 1 provides the data for the demand, disassembly direction and disassembly method required for each component. We assume that $n = 10$.

Table 1. Initial Data for Numerical Example

Item	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
Demand	No	Yes	Yes	No	Yes	Yes	Yes	No	No	No
Direction	$+y$	$+x$	$+x$	$+y$	$-z$	$-z$	$+y$	$-x$	$-z$	$-y$
Method	*D*	*N*	*N*	*D*	*N*	*N*	*D*	*D*	*N*	*D*

Using the above data, an example of the initial population created randomly is as follows:

Table 2. Initial Population

#	Chromosome
0	2769431508, *+x-y+y-y-z+y+x-z+y-z,NDDDNDNNDN*
1	1732860594, *+x-y+y+x-z+y+y-z-y-z,NDDNNDDNDN*
2	1782649053, *+x-y-z+x+y-z-y+y-z+y,NDNNDNDDND*
3	1764935802, *+x-y+y-z-y+y-z-z+y+x,NDDNDDNNDN*
4	2765938410, *+x-y+y-z-y+y-z-z+x+y,NDDNDDNNND*
5	1279360845, *+x+x-y-y+y+y+y-z-z-z,NNDDDDDNNN*
6	2073869514, *+x+y-y+y-z+y-y-z+x-z,NDDDNDDNNN*
7	2761503948, *+x-y+y+x-z+y+y-y-z-z,NDDNNDDDNN*
8	2783694510, *+x-y-z+y+y-y-z-z+x+y,NDNDDDNNND*
9	1876429503, *+x-z-y+y-z+x-y-z+y+y,NNDDNNDNDD*

3. 2. Genetic Operators

Crossover

We use the precedence preservative crossover (PPX) methodology for this operation. In this methodology, in addition to the two strings representing the chromosomes of the parents, two additional strings representing operators for crossover are added. The operators pass on the precedence relationship based on the two parental permutations to two new offspring while making sure that no new precedence relationships are introduced. A vector of length *i*, representing the number of operations involved in the problem, is randomly filled with elements of the set. This vector defines the order in which the operations are successively drawn from parent 1 and parent 2. First we start by initializing an empty offspring. The leftmost operation in one of the two parents is selected in accordance with the order of parents given in the vector. After an operation is selected it is deleted in both parents. Finally the selected operation is appended to the offspring. This step is repeated until both parents are empty and the offspring contains all operations involved.

As an example, consider two chromosomes, parent 1 and parent 2. The strings of these chromosomes are as follows:

Parent 1: 2 1 9 7 6 0 8 5 3 4, $+x\ +x\ -y\ -x\ +y\ +y\ -z\ -z\ +y\ -z$, $N\ N\ D\ D\ D\ D\ N\ N\ D\ N$
Parent 2: 1 0 7 2 6 8 4 9 3 5, $+x\ +y\ -x\ +x\ +y\ -z\ -z\ -y\ +y\ -z$, $N\ D\ D\ N\ D\ N\ N\ D\ D\ N$

Since the offspring depends on only the first part of the parents' chromosomes, we can simplify the two strings as follows:

Parent 1: 2 1 9 7 6 0 8 5 3 4
Parent 2: 1 0 7 2 6 8 4 9 3 5

Next we create the mask arrays for the two children. The masks consist of numbers 1 and 2 representing the parent number from which the gene is selected. Assume that the two random masks for child 1 and child 2 are as follows:

Mask for child 1: 2 2 2 1 1 2 2 2 1 2
Mask for child 2: 2 1 1 2 1 2 2 1 2 2

Then, the first part of the chromosomes of child 1 and child 2 become:

Child 1: 1 0 7 2 9 6 8 4 5 3
Child 2: 1 2 9 0 7 6 8 5 4 3

Thus the chromosomes of child 1 and child 2 can be written as:

Child 1: 1 0 7 2 9 6 8 4 5 3, $+x\ +y\ -x\ +x\ -y\ +y\ -z\ -z\ -z\ +y$, $N\ D\ D\ N\ D\ D\ N\ N\ N\ D$
Child 2: 1 2 9 0 7 6 8 5 4 3, $+x\ +x\ -y\ +y\ -x\ +y\ -z\ -z\ -z\ +y$, $N\ N\ D\ D\ D\ D\ N\ N\ N\ D$

Mutation

The chromosomes mutate with a given probability. The mutation operator selects a random number of genes, and exchanges them in such a way that the same precedence relationship is maintained. After the crossover operation the population is sent to mutation operation. As an example consider the following:

Population		
#	Before Mutation	After Mutation
0	1276394805	2176395804
1	2176483095	1276583094
2	1273694085	2173695084
3	1273649058	2173659048
4	2176483095	2176483095
5	1273690458	1273690458
6	2176483095	2176483095
7	1273690458	1273690458
8	1276390458	1276390458
9	1276390485	1276390485

As it can be seen from the new population only the first four (predetermined number) chromosomes are selected for mutation. The rest of the strings remain the same. This is done to preserve some of the strong solutions in the current population.

3. 3. Fitness Evaluation

There are three goals to be considered in disassembly process planning as described below:

i. *Disassembling the demanded items as soon as possible:* In our example components 1-2-4-5-6 are demanded. In calculating the fitness of a demanded component, a penalty is levied on each delay and an award is given for early disassembly. A similar logic also holds for the non-demanded components. Algorithmically, the partial fitness can be given as below.

if (gene $\in$ {demand set}) $\Rightarrow$
Penalty = (-1) . (index of gene) ^{2}
Award = (chrom length – index of gene)^2
Partial Fitness for demanded items = Penalty + Award
= chrom length^2 – (2 . chrom length . index of gene) + index of gene^2 – index of gene^2
= chrom length^2 – (2 . chrom length . index of gene)
else if (gene $\notin$ {demand set}) $\Rightarrow$
Penalty = (-1) . (chrom length – index of gene)^2
Award = (index of gene) ^{2}
Partial Fitness for non-demanded items = Penalty + Award
= index of gene^2 – index of gene^2 – chrom length^2 + (2 . chrom length . index of gene)
= (2 . chrom length . index of gene) – chrom length^2

Note that, since genes in a chromosome are indexed starting from zero, if a demanded gene is first in the sequence, it will incur no penalty. The fitness function supports the demanded components as early as possible and the non-demanded items to be disassembled as late as possible. Note also that the chrom length (chromosome length) refers to only the first part (one third) of a given chromosome, which consists of component numbers from 0 to 9.

ii. *Decreasing the number of directional changes:* Since the directional changes increase the setup time of the disassembly process, each directional change is penalize by a negative constant number (chosen appropriately). The award is also given in a similar manner. Thus,

Penalty = number of "directional changes" . penalty constant for directional change
Award = number of "no directional changes" . award constant for no directional change
Partial fitness for directional changes = Award + Penalty

iii. *Decreasing the number of disassembly method changes:* Grouping the same type of disassembly method (viz. destructive or non-destructive) will help reduce the setup time for disassembly. Thus, each disassembly method change is penalized by a negative constant number (chosen appropriately). A similar philosophy is adopted for the award. Thus:

Penalty = number of "disassembly method changes" . penalty constant for disassembly method change
Award = number of "no disassembly method changes". award constant for stable disassembly method
Partial fitness for disassembly method changes = Award + Penalty

3. 4. An Illustration for Fitness Calculation

As an illustration for the fitness calculation, consider a random chromosome in our example:

Chromosome: 1 0 7 2 9 6 8 4 5 3, $+x\ +y\ -x\ +x\ -y\ +y\ -z\ -z\ -z\ +y$, $N\ D\ D\ N\ D\ D\ N\ N\ N\ D$

The partial fitness for the demand:
The partial fitness for the demand can be calculated as follows:

Index: 0 1 2 3 4 5 6 7 8 9
String: 1 0 7 2 9 6 8 4 5 3
Partial Fitness for demand = 100 – 2 . 10 . (0) + 100 – 2 . 10 . (3) + 100 – 2 . 10 . (8) +
100 – 2 . 10 . (7) + 100 – 2 . 10 . (5) + 2 . 10 . (1) – 100 + 2 . 10 . (2) – 100 –
2 . 10 . (4) – 100 – 2 . 10 . (6) – 100 – 2 . 10 . (9) – 100
= 100 + 40 – 60 – 40 + 0 – 80 – 60 – 20 + 20 + 80
= –20

The partial fitness for the directional change:
Assuming the value of the constant number to be 50, the partial fitness for directional changes can be calculated as follows:

Penalty = number of "directional changes" . penalty for directional change
= 7 * (–50) = – 350
Award = number of "no directional changes". award for no directional change
= 2 * (50) = 100
Partial fitness for directional changes
= 100 + (– 350) = – 250.

The partial fitness for the disassembly method change:
Again, assuming the value of the constant number to be 50, the partial fitness for disassembly method changes can be calculated as follows:

Penalty = number of "disassembly method changes" . penalty for disassembly method change
= 5 * (–50) = – 250
Award = number of "no disassembly method changes" . award for no disassembly method change
= 4 * (50) = 200
Partial fitness disassembly method changes
= 200 + (– 250) = – 50

Hence, the total fitness of the chromosome is the sum of all the fitness values as follows:

Fitness of the chromosome
= (– 20) + (– 250) + (– 50) = – 320

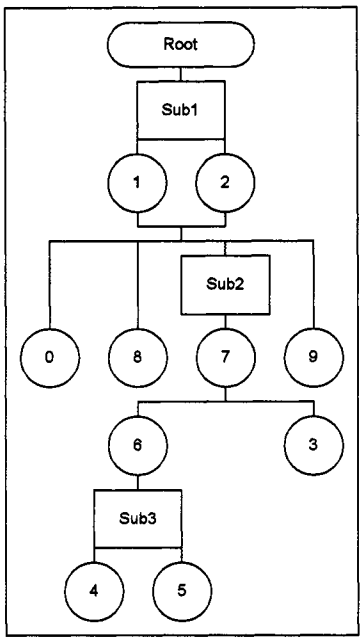
3. 5. Selection Procedure

After every generation, the chromosomes in the current population are sorted according to their fitness values. The first half of the sorted population is selected and cloned. This way, a new population is generated eliminating the weak chromosomes. This method seeks to improve the population while allowing the better chromosome generation with further string manipulations such as crossover and mutation.

4. CASE EXAMPLE

Consider the product in Figure 1 and the data as given in Table 1. The crossover and mutation probabilities are assumed to be 0.60 and 0.005 respectively.

Figure 1. Original Product Structure



The initial population of 10 chromosomes is randomly generated and is as shown in Table 3.

Table 3. Initial Population for the Numerical Example

#	Chromosome String	Fitness
0	1720683459, *+x–x+x+y+y–z+y–z–z–y,NDNDDNDNND*	-1490
1	1762584903, *+x–x+y+x–z–z–z–y+y+y,NDDNNNNDDD*	-750
2	2763198504, *+x–x+y+y+x–y–z–z+y–z,NDDDNDNNDN*	-1430
3	2731650984, *+x–x+y+x+y–z+y–y–z–z,NDDNDNDDNN*	-1490
4	1763429058, *+x–x+y+y–z+x–y+y–z–z,NDDDNNDDNN*	-1110
5	1723098645, *+x–x+x+y+y–y–z+y–z–z,NDNDDDNDNN*	-1590
6	2176450839, *+x+x–x+y–z–z+y–z+y–y,NNDDNNDNDD*	-970
7	1076932458, *+x+y–x+y–y+y+x–z–z–z,NDDDDDNNNN*	-1110
8	1736908425, *+x–x+y+y–y+y–z–z+x–z,NDDDDDNNNN*	-1230
9	1876350924, *+x–z–x+y+y–z+y–y+x–z,NNDDDNDDNN*	-1450

A C-programming language code was written that was adapted from the basic genetic algorithm code provided by Michalewicz (1996). The execution of the code ceases if one of the following two conditions is met. 1) The number of generations reaches up to a maximum value (100 in our example), or 2) the ratio of the average fitness value of the new population to the average fitness value of the old population has a value less than or equal to a certain number (1.0005 in our example).

After the generation of six populations the solution of our example is obtained in a negligible amount of execution time. The final population is as shown in Table 4.

Table 4. Final Population for the Numerical Example

#	Chromosome String	Fitness
0	2176458039, *+x+x–x+y–z–z–z+y+y–y,NNDDNNNDDD*	430
1	2176548039, *+x+x–x+y–z–z–z+y+y–y,NNDDNNNDDD*	430
2	2176458039, *+x+x–x+y–z–z–z+y+y–y,NNDDNNNDDD*	430
3	2176548039, *+x+x–x+y–z–z–z+y+y–y,NNDDNNNDDD*	430
4	2176548039, *+x+x–x+y–z–z–z+y+y–y,NNDDNNNDDD*	430
5	2176458039, *+x+x–x+y–z–z–z+y+y–y,NNDDNNNDDD*	430
6	2176548039, *+x+x–x+y–z–z–z+y+y–y,NNDDNNNDDD*	430
7	2176458039, *+x+x–x+y–z–z–z+y+y–y,NNDDNNNDDD*	430
8	1276548039, *+x+x–x+y–z–z–z+y+y–y,NNDDNNNDDD*	430
9	2176548039, *+x+x–x+y–z–z–z+y+y–y,NNDDNNNDDD*	430

As can be seen in Table 4, the fitness values of all the chromosomes are the same (430). That means, they are all equally desirable. However, there are only two distinct chromosomes (or disassembly sequences) in the population, viz., 2,1,7,6,4,5,8,0,3,9, *+x, +x,–x, +y,–z,–z,–z, +y, +y,–y,N,N,D,D,N,N,N,D,D,D* and 1,2,7,6,5,4,8,0,3,9, *+x, +x,–x, +y,–z,–z,–z, +y, +y,–y,N,N,D,D,N,N,N,D,D,D.*

5. CONCLUSIONS

A genetic algorithm model was presented in order to determine the optimal disassembly sequence of a given product. For the example considered, the algorithm provided optimal disassembly sequence in a short execution time. The algorithm is practical, as it is easy to use, considers the precedence relationship and additional constraints in the product structure and is applicable to problems with multi-objectives.

REFERENCES

1. Bierwirth C. , Mattfeld D. C., and Kopfer H., "On permutation representations for scheduling problems," in Parallel Problem Solving from Nature -- PPSN IV, H.-M. Voigt, W. Ebeling, I. Rechenberg, and H.-P. Schwefel, eds., vol. 1141 of Lecture Notes in Computer Science, pp. 310--318, Springer-Verlag, Berlin, Germany, 1996.
2. Bierwirth C. and Mattfeld D. C., Production scheduling and rescheduling with genetic algorithms, *Evolutionary Computation*, 7(1):1--18, 1999.
3. Blanton, J. L. and Wainwright, R. L., Multiple vehicle routing with time and capacity constraints using genetic algorithms. In Forrest, S., editor, *Proceedings of the Fifth International Conference on Genetic Algorithms*, pages 452–459, Morgan Kaufmann, San Mateo, California, 1993.
4. Gungor, A. and Gupta, S. M., "Issues in Environmentally Conscious Manufacturing and Product Recovery: A Survey", *Computers and Industrial Engineering*, Vol. 36, No. 4, 811-853, 1999.
5. Kuo T.C., "Disassembly Sequence and Cost Analysis for Electromechanical Products", *Robotics and Computer Integrated Manufacturing*, Vol. 16, No. 1, 43-54, 2000.
6. Lazzerini B., Marcelloni F., "A genetic algorithm for generating optimal assembly plans", *Artificial Intelligence in Engineering*, Vol. 14, 319-329, 2000.
7. Lee, D.-H., Kang, J.-G. and Xirouchakis, P., "Disassembly Planning and Scheduling: Review and Further Research", *Journal of Engineering Manufacture*, Vol. 215, No. B5, 695-709, 2001
8. Loughlin D. H., Ranjithan S., The neighborhood constraint-method: A genetic algorithm-based multiobjective optimization technique, *Proceedings of the Seventh International Conference on Genetic Algorithms*, 666-673, 1997.
9. Michalewicz, Z., "Genetic Algorithms + Data Structures = Evolution Programs", 3rd edition, Springer-Verlag, Berlin, 1996.

10. Moore, K. E., Gungor, A. and Gupta, S. M., "Petri Net Approach to Disassembly Process Planning for Products with Complex AND/OR Precedence Relationships", *European Journal of Operational Research*, Vol. 135, No. 2, 189-210, 2001.
11. Moyer, L. K. and Gupta, S. M., "Environmental concerns and Recycling/Disassembly Efforts in the electronic industry", *Journal of Electronics Manufacturing*, 7(1), 1-22, 1997.
12. Onwubolu G. C., and Mutingi, M., "Optimizing the multiple constrained resources product mix problem using genetic algorithms", *International Journal of Production Research*, 39(9), 1897-1910, 2001.
13. Ritchey J., Mahmoodi F., Frascatore M., Zander A., "Assessing the Technical and Economic Feasibility of Remanufacturing", *Proceedings of the Twelfth Annual Conference of the Production and Operations Management Society, POM-2001*, March 30-April 2, Orlando, FL, 2001.
14. Sanderson A. C., Homem de Mello L. S., Zhang H., "Assembly sequence planning", *AI Magazine*, Spring 1990, 62-81, 1990.
15. Seo K.-K., Park J.-H. and Jang D.-S., "Optimal Disassembly Sequence Using Genetic Algorithms Considering Economic and Environmental Aspects", *International Journal of Advanced Manufacturing Technology*, Vol. 18, 371–380, 2001.
16. Taleb, K. and Gupta, S. M., "Disassembly of Multiple Product Structures", *Computers and Industrial Engineering*, Vol. 32, No. 4, 949-961, 1997.
17. Tang, Y., Zhou, M., Zussman, E., and Caudill, R., "Disassembly Modeling, Planning, and Application: A Review", *Proceedings of the 2000 IEEE International Conference on Robotics and Automation,* San Francisco, California, April, pp. 2197-2202, 2000.
18. Valenzuela-Rendón M, Uresti-Charre E., "A non-generational genetic algorithm for multiobjective optimization" *Proceedings of the Seventh International Conference on Genetic Algorithms*, 658-665, 1997.
19. Veerakamolmal, P., Gupta, S. M. and McLean C. R., "Disassembly Process Planning", *Proceedings of the First Int. Conference on Engineering Design and Automation*, Bangkok, Thailand, 162-165, March 18-21, 1997.
20. Wong L., "Barriers to implementation of electronic end of life product programs in the U.S., Investigating the Environment: Research for Environmental Management, *Compilation of Research Reports from the Fall 2000/Spring 2001 Senior Research Seminars*, Environmental Sciences Group Major, University of California at Berkeley, Berkeley, California, May 2001.